ИНФОРМАТИКА

2010 **январь-март** **№ 1**

ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

УДК 519.7

А.Д. Закревский, Н.Р. Торопов, В.И. Романов

ДНФ-РЕАЛИЗАЦИЯ ЧАСТИЧНЫХ БУЛЕВЫХ ФУНКЦИЙ МНОГИХ ПЕРЕМЕННЫХ

Предлагается эвристический метод минимизации частичных булевых функций многих переменных, ориентированный на компьютерную реализацию. В его основе лежит оригинальная технология параллельных операций над соседними элементами в булевом пространстве многих переменных. Приводятся результаты программной реализации метода и экспериментов на компьютере, в ходе которых оценивались эффективность программы и границы ее применимости.

Введение

Задача минимизации булевых функций в классе дизъюнктивных нормальных форм (ДНФ) принадлежит к числу наиболее известных проблем теории булевых функций и имеет разнообразные практические применения. Ей посвящено не менее тысячи публикаций, список которых открывается классическими работами Квайна [1] и Мак-Класки [2]. Построение строго минимальной ДНФ для произвольной булевой функции от n переменных является трудной комбинаторной задачей, практически решаемой лишь при относительно малом n (максимум 12). Поэтому широкое распространение получили приближенные методы, позволяющие находить на компьютере близкие к оптимальным решения за приемлемое время [3]. Однако с ростом числа переменных n их трудоемкость стремительно возрастает, в связи с чем появилась необходимость в разработке новых методов, применимых при большом числе переменных (20 и выше). Один из таких методов был предложен в работах [4–6] для минимизации полностью определенных булевых функций. Он послужил базой для разработки эвристического метода минимизации *частичных* (не везде определенных) булевых функций многих переменных, впервые описываемого в настоящей статье.

1. Векторное представление булевых функций

Любая полностью определенная булева функция $f(\boldsymbol{x}) \equiv f(x_1, x_2, \ldots, x_n)$ может быть задана булевым 2^n-вектором $\boldsymbol{f}$ с 2^n компонентами, пронумерованными слева направо, начиная с нуля, и поставленными в соответствии с элементами n-мерного булева пространства $M = \{0, 1\}^n$. При этом компонента с номером k задает значение функции f на наборе значений аргументов, представляющем общепринятый двоичный код $\boldsymbol{b}^k$ числа k. Совокупность отмеченных единицами элементов пространства M образует характеристическое множество M^1 функции f.

Например, следующий вектор $\boldsymbol{f}$ задает булеву функцию пяти переменных $x_1, x_2, \ldots, x_5$:

```
                        -------- --------     1
              --------           --------    2
      ----      ----      ----      ----     3
    --  --    --  --    --  --    --  --    4
  - - - -   - - - -   - - - -   - - - -   5
```

10010101 01100110 00101101 10110010 = *f*

Единицами вектор отмечает 16-элементное характеристическое множество {00000, 00011, 00101, …, 11110}, и его можно интерпретировать как совершенную нормальную форму булевой функции f, используемую в классических методах минимизации булевых функций. Заметим, что ее непосредственное представление оказывается существенно более громоздким:

$$f = \bar{x}_1\,\bar{x}_2\,\bar{x}_3\,\bar{x}_4\,\bar{x}_5 \vee \bar{x}_1\,\bar{x}_2\,\bar{x}_3\,x_4\,x_5 \vee \bar{x}_1\,\bar{x}_2\,x_3\,\bar{x}_4\,x_5 \vee \ldots \vee x_1\,x_2\,x_3\,x_4\,\bar{x}_5.$$

При числе переменных, превышающем 5, вектор $\boldsymbol{f}$ удобно представлять в виде булевой матрицы размером 2^{n-5} на 2^5, отображая ее 32-компонентные строки словами в компьютерной памяти (что соответствует организации памяти большинства современных компьютеров). Тогда любые два элемента вектора $\boldsymbol{f}$, соседние по переменной x_i, будут принадлежать одному слову, если $i < 6$, и к разным словам в противном случае, что можно использовать для ускорения вычислений.

Частичная булева функция задается одним троичным вектором либо двумя булевыми, что более удобно при программировании. Во втором случае обычно используются два булевых вектора $\boldsymbol{f}^1$ и $\boldsymbol{f}^0$, представляющие соответственно характеристические множества M^1 (на котором $f = 1$) и M^0 ($f = 0$).

В настоящей статье описывается метод, предназначенный для минимизации произвольных булевых функций со следующими параметрами: n – число переменных, r – плотность единиц, s – степень неопределенности. Они задают с точностью до 1/32 ожидаемые относительные числа: $r/32$ единичных значений в полностью определенной функции и относительное число $s/32$ неопределенных значений в частичной функции. Например, если $r = 16$, рассматривается совершенно случайная булева функция. Такие функции получаются автоматически в экспериментальной системе, используемой при оценке эффективности реализующей данный метод программы и определении границ ее практической применимости. Базисной операцией при этом служит получение квазислучайного булева вектора $\boldsymbol{p}$ с 2^n компонентами, математическое ожидание числа единиц в котором определяется параметром p и равно $2^n p/32$. Вектор $\boldsymbol{p}$ находится по методу, предложенному в [7] и использующему последовательность ($\boldsymbol{q}_1$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$, $\boldsymbol{q}_4$, $\boldsymbol{q}_5$) случайных булевых 2^n-векторов с равномерным распределением единиц и нулей.

Проиллюстрируем этот метод следующим примером: пусть $p = 25$, выразим это число в двоичном пятикомпонентном коде 11001, присвоим для начала вектору $\boldsymbol{p}$ значение $\mathbf{0}$, а затем выполним пять покомпонентных логических операций $\boldsymbol{q}_1 \vee (\boldsymbol{q}_2 \vee (\boldsymbol{q}_3 \wedge (\boldsymbol{q}_4 \wedge (\boldsymbol{q}_5 \vee \mathbf{0}))))$, поставив в соответствие компоненте 1 кода оператор $\vee$, а компоненте 0 – оператор $\wedge$. В результате получим случайный булев 2^n-вектор, ожидаемое число единиц в котором равно 25/32.

Описываемый в статье метод минимизации булевых функций основан на применении эффективных параллельных операций над булевыми 2^n-векторами [8, 9]. К таким операциям относится, в частности, операция конъюнктивного симметрирования вектора $\boldsymbol{f}$ по переменной x_i, обозначаемая далее через $\mathrm{S}\boldsymbol{f} \wedge i$. При ее выполнении вектор $\boldsymbol{f}$ разбивается на 2^{n-1} пар компонент, соседних по переменной x_i, и оба элемента каждой пары получают значение, равное конъюнкции исходных значений этих элементов. Напомним, что *соседними* называются такие компоненты вектора $\boldsymbol{f}$, которым соответствуют наборы значений аргументов, различающиеся ровно в одном аргументе.

2. Выявление перспективных элементов в характеристическом множестве

Приближенные алгоритмы минимизации полностью определенной булевой функции f обычно начинаются с поиска таких элементов множества M^1, у которых мало соседей в этом же множестве (эти элементы будем называть перспективными). При большой мощности множества M^1 данная процедура оказывается довольно трудоемкой, включая рассмотрение многочисленных комбинаций элементов из M^1. Использование операции $\mathrm{S}\boldsymbol{f} \wedge i$ существенно ускоряет поиск. С помощью данной операции строится булева *матрица соседства* $\boldsymbol{N}$ размером $n \times 2^n$ со строками $\boldsymbol{n}_i = \mathrm{S}\boldsymbol{f} \wedge i$. Матрица $\boldsymbol{N}$ отображает структуру характеристического множества M^1 функции $f(\boldsymbol{x})$. Элемент n_i^k матрицы $\boldsymbol{N}$ принимает значение 1, если и только если k-й элемент вектора $\boldsymbol{f}$ равен 1 и имеет соседа по переменной x_i также со значением 1.

Продемонстрируем получение матрицы $\boldsymbol{N}$ на примере вектора

10010101 01100110 00101101 10110010 = $\boldsymbol{f}$.

Пятикратное применение операции $S\boldsymbol{f}\wedge i$ приводит к следующему результату:

$$\boldsymbol{N}=\begin{array}{ll}
00000101\ 00100010\ 00000101\ 00100010 & = S\boldsymbol{f}\wedge 1\\
00000100\ 00000100\ 00100000\ 00100000 & = S\boldsymbol{f}\wedge 2\\
00010001\ 01100110\ 00000000\ 00100010 & = S\boldsymbol{f}\wedge 3\\
00000101\ 00000000\ 00000101\ 10100000 & = S\boldsymbol{f}\wedge 4\\
00000000\ 00000000\ 00001100\ 00110000 & = S\boldsymbol{f}\wedge 5
\end{array}.$$

Поиск перспективных элементов облегчается сортировкой элементов множества M^1 по числу соседей, результат которой представляется булевыми векторами $\boldsymbol{m}_i$. Единицами в них отмечены элементы множества M^1, число соседей которых равно i. Эти векторы легко получить покомпонентными операциями над строками матрицы $\boldsymbol{N}$. Для данного примера

$\boldsymbol{m}_0$ = 10000000 00000000 00000000 00000000;
$\boldsymbol{m}_1$ = 00010000 01000000 00101000 10010000;
$\boldsymbol{m}_2$ = 00000000 00100110 00000001 00000010;
$\boldsymbol{m}_3$ = 00000101 00000000 00000100 00000000 и т. д.

3. Матричная форма решения

Получаемую в результате минимизации ДНФ предлагается представлять в форме, аналогичной вектору функции $\boldsymbol{f}$ и матрице соседства $\boldsymbol{N}$, булевым вектором $\boldsymbol{g}$ и булевой матрицей $\boldsymbol{D}$ той же размерности, задающими некоторую совокупность интервалов булева пространства $M=\{0,1\}^n$. В векторе $\boldsymbol{g}$ отмечаются некоторые содержащиеся в интервалах элементы множества M^1 (по одному для каждого интервала), а в столбцах матрицы $\boldsymbol{D}$ отмечаются внутренние переменные этих же интервалов.

Рассматривая векторы $\boldsymbol{f}$ и $\boldsymbol{g}$ как множества отмеченных в них элементов пространства M, а матрицы $\boldsymbol{N}$ и $\boldsymbol{D}$ – как подмножества декартова произведения множеств $\boldsymbol{x}$ и M, сформулируем очевидное

Утверждение 1. $\boldsymbol{g}\subseteq\boldsymbol{f}$ *и* $\boldsymbol{D}\subseteq\boldsymbol{N}$.

Другими словами, вектор решения $\boldsymbol{g}$ и матрица решения $\boldsymbol{D}$ могут быть получены соответственно из вектора $\boldsymbol{f}$ и матрицы $\boldsymbol{N}$ заменой в них некоторых единиц на нули.

Для рассматриваемого примера искомая ДНФ будет выглядеть следующим образом:

$$\boldsymbol{g}=10010000\ 01100000\ 00101001\ 10010000;$$

$$\boldsymbol{D}=\begin{array}{l}
00000000\ 00100000\ 00000001\ 00000000\\
00000000\ 00000000\ 00100000\ 00000000\\
00010000\ 01100000\ 00000000\ 00000000\\
00000000\ 00000000\ 00000001\ 10000000\\
00000000\ 00000000\ 00001000\ 00010000
\end{array}.$$

В более известной форме эта ДНФ задается троичной матрицей, столбцы которой представляют элементарные конъюнкции ($\bar{x}_1\,\bar{x}_2\,\bar{x}_3\,\bar{x}_4\,\bar{x}_5$, $\bar{x}_1\,\bar{x}_2\,\bar{x}_3\,x_4\,x_5$ и т. д.)

$$\begin{array}{ll}
x_1 & 0\ 0\ 0\ -\ 1\ 1\ -\ 1\ 1\\
x_2 & 0\ 0\ 1\ 1\ -\ 0\ 0\ 1\ 1\\
x_3 & 0\ -\ -\ -\ 0\ 1\ 1\ 0\ 0\\
x_4 & 0\ 1\ 0\ 1\ 1\ 0\ -\ -\ 1\\
x_5 & 0\ 1\ 1\ 0\ 0\ -\ 1\ 0\ -
\end{array}$$

Число p конъюнкций в полученной ДНФ равно весу вектора $\boldsymbol{g}$, а длина ДНФ (сумма рангов конъюнкций) равна $pn-q$, где q – число единиц в матрице $\boldsymbol{D}$.

4. Нахождение простых импликант

Построение реализующей функцию f ДНФ целесообразно начинать с поиска элементов ядра решения – обязательных простых импликант высокого ранга. Назовем *обязательной* такую простую импликанту функции f, которая входит в любую кратчайшую ДНФ этой функции.

Обозначим через $\boldsymbol{t}^k$ троичный вектор, получаемый из вектора $\boldsymbol{b}^k$ присвоением значения «–» компонентам, отмеченным единицами в столбце $\boldsymbol{n}^k$ матрицы $\boldsymbol{N}$. Его можно интерпретировать как некоторый интервал Int_k пространства $M = \{0, 1\}^n$ и как соответствующую элементарную конъюнкцию, которая может оказаться простой импликантой функции f.

Утверждение 2. *Вектор $\boldsymbol{t}^k$ представляет обязательную простую импликанту функции f, если и только если $Int_k \subseteq M^1$.*

Утверждение 3. *Для каждой обязательной простой импликанты функции f в матрице $\boldsymbol{N}$ найдется столбец $\boldsymbol{n}^k$, которому соответствует троичный вектор $\boldsymbol{t}^k$, представляющий эту импликанту.*

Легко находятся обязательные простые импликанты ранга n, непосредственно представляемые элементами множества M^1, которые не имеют соседей. Они отмечаются в векторе $\boldsymbol{m}_0$ и задаются соответствующими столбцами матрицы $\boldsymbol{N}$, не содержащими единиц. Аналогично все обязательные простые импликанты ранга $n - 1$ отмечены в векторе $\boldsymbol{m}_1$ и также представлены соответствующими столбцами матрицы $\boldsymbol{N}$ – они содержат по одной единице. Выявление обязательных простых импликант меньшего ранга происходит несколько сложнее.

Утверждение 4. *Предположим, что k-й элемент вектора $\boldsymbol{f}$ имеет двух соседей. Тогда вектор $\boldsymbol{t}^k$ представляет обязательную простую импликанту функции f, если и только если j-я компонента вектора $\boldsymbol{f}$ равна единице, при этом $\boldsymbol{b}^j = \boldsymbol{b}^k \oplus \boldsymbol{n}^k$.*

Например, $\boldsymbol{b}^{10} \oplus \boldsymbol{n}^{10} = 01010 \oplus 10100 = 11110$, $j = 30$ и $f_{30} = 1$, следовательно, троичный вектор $\boldsymbol{t}^{10} = –1–10$ представляет обязательную простую импликанту. Ее можно внести в решение, а покрываемые ею единицы исключить из вектора $\boldsymbol{f}^*$, который отмечает пока непокрытые элементы множества M^1.

Утверждение 5. *Предположим, что k-й элемент вектора $\boldsymbol{f}$ имеет трех соседей. Тогда вектор $\boldsymbol{t}^k$ представляет обязательную простую импликанту функции f, если и только если $\boldsymbol{n}^k \le \boldsymbol{n}^j$ (вектор $\boldsymbol{n}^k$ поглощается вектором $\boldsymbol{n}^j$), при этом $\boldsymbol{b}^j = \boldsymbol{b}^k \oplus \boldsymbol{n}^k$.*

5. Построение частичного решения

В предлагаемом алгоритме последовательно находятся импликанты ранга n, n–1, n–2 и n–3. Результат фиксируется введением единиц в вектор решения $\boldsymbol{g}$ (сначала $\boldsymbol{g} = \boldsymbol{0}$), определением соответствующих им столбцов матрицы $\boldsymbol{D}$ и корректировкой вектора $\boldsymbol{f}^*$, в котором отмечаются непокрытые элементы множества M^1 (сначала $\boldsymbol{f}^* = \boldsymbol{f}$).

Импликанты ранга n и n–1 определяются однозначно при наличии в множестве M^1 соответствующих элементов, имеющих не более одного соседа. Так в рассматриваемом примере находятся импликанты 00000, 00–11, 01–01, 1–010, 1010–, 110–0, 1101–, и вектор решения принимает значение

$$\boldsymbol{g} = \texttt{10010000 01000000 00101000 10010000}.$$

Соответственно меняется значение вектора $\boldsymbol{f}^*$:

$$\boldsymbol{f}^* = \texttt{00000100 00100010 00000001 00000010}.$$

Далее рассматриваются последовательно элементы f^k с двумя соседями, отмеченные одновременно в векторах $\boldsymbol{m}_2$ и $\boldsymbol{f}^*$. Сначала находятся элементы f^k, удовлетворяющие условию утверждения 4. Соответствующие компоненты вектора $\boldsymbol{g}$ принимают значение 1, столбцы $\boldsymbol{d}^k$ матрицы $\boldsymbol{D}$ остаются равными столбцам $\boldsymbol{n}^k$ матрицы $\boldsymbol{N}$, и из вектора $\boldsymbol{f}^*$ удаляются единицы, покрываемые интервалами Int_k. Если же условие утверждения 4 не выполняется, из двух соседей элемента f^k выбирается один сосед, желательно еще не покрытый, в столбце $\boldsymbol{d}^k$ остается лишь соответствующая единица и выполняется операция, предусмотренная для элемента с одним соседом.

Последними рассматриваются элементы с тремя соседями, отмеченные одновременно в векторах $\boldsymbol{m}_3$ и $\boldsymbol{f}^*$. Среди них находятся элементы, удовлетворяющие условию утверждения 5. Они обрабатываются так же, как и элементы с двумя соседями, удовлетворяющие условию утверждения 4. В результате выявляются импликанты ранга $n-3$. При невыполнении указанного условия для рассматриваемого элемента находится некоторая импликанта ранга $n-2$ или $n-1$.

В результате описанной процедуры обработки элементов вектора $\boldsymbol{f}$, имеющих не более трех соседей, получается множество импликант, образующих *частичное решение S*, и вектор $\boldsymbol{f}^*$, представляющий *остаток* – совокупность непокрытых этим решением элементов множества M^1. В данном примере (он оказывается довольно простым) эта процедура приводит к окончательному решению задачи минимизации полностью определенной булевой функции: покрытию всех элементов характеристического множества M^1 и получению вектора решения $\boldsymbol{g}$ и матрицы решения $\boldsymbol{D}$, описанных в разд. 3.

6. Итеративный алгоритм поиска импликант частичной булевой функции

Рассмотренная процедура получения частичного решения применима и в приложении к частичной булевой функции, заданной двумя характеристическими множествами M^1 и M^0. В этом случае рассматривается лишь множество M^1, представленное булевым вектором $\boldsymbol{f} = \boldsymbol{f}^1$, а множество M^0 временно игнорируется. Полученное частичное решение полностью определенной булевой функции принимается за частичное решение частичной функции.

Идея итеративного алгоритма состоит в следующем. Сначала он находит частичное решение для вектора $\boldsymbol{f}$, затем выполняет такую же операцию для остатка $\boldsymbol{f}^*$, дополняя множество получаемых импликант и соответственно упрощая вектор $\boldsymbol{f}^*$ (удаляя из него некоторые единицы). Если после упрощения $\boldsymbol{f}^*$ в нем остаются единицы, выполняется следующая итерация и т. д., пока $\boldsymbol{f}^*$ не станет равным нулю.

Алгоритм может прекратить свою работу, если после очередной итерации в векторе остаются единичные элементы, но число соседей в каждом из них превышает 3. Эта возможность рассмотрена в [4], где экспериментально определены условия ее реализации для случая полностью определенной булевой функции. Заметим, что при минимизации частичных булевых функций такая ситуация практически не возникает.

Продемонстрируем алгоритм на конкретном примере полностью определенной функции ($s = 0$). Пусть $n = 15$ и каждый элемент вектора $\boldsymbol{f}$ принимает значение 1 с вероятностью ½ ($r = 16$).

Для этого примера выполняется двенадцать итераций, прежде чем вектор $\boldsymbol{f}^*$ становится равным 0. Все элементы множества M^1 оказываются покрытыми 5339 импликантами со следующим распределением их по рангам (ранг – число символов переменных в импликанте):

$$15 - 1,\ 14 - 150,\ 13 - 3261,\ 12 - 1902,\ 11 - 25.$$

Дизъюнкция полученных импликант образует ДНФ (длиной 67 607) рассматриваемой булевой функции. Однако она не является безызбыточной, поскольку импликанты могут быть не простыми и некоторые из них можно без ущерба удалить из ДНФ. Соответствующая операция устранения избыточности описывается в следующем разделе. Она оказывается существенно более трудоемкой в сравнении с итеративным алгоритмом, зато значительно упрощает ДНФ. В данном примере полученное решение приводится к более компактной форме, содержащей 4816 импликант и образующей ДНФ длиной 61 088.

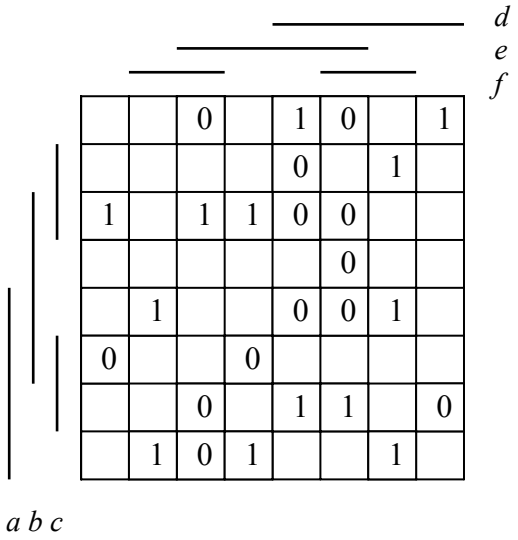
При рассмотрении частичной булевой функции выигрыш от устранения избыточности в формуле может быть значительно выше, поскольку такая функция реализуется многочисленными полностью определенными функциями с ДНФ различной сложности.

7. Приведение решения к безызбыточной форме

При построении безызбыточной формы прежде всего упрощается каждый член ДНФ, полученной итеративным алгоритмом. Последовательно рассматриваются образующие его символы переменных. Очередной символ удаляется, если интервал пространства M, соответст-

вующий получаемой при этом конъюнкции, не пересекается с множеством M^0. Затем сокращается длина ДНФ путем удаления некоторых импликант, поглощаемых совокупностью остающихся (имплицирующих их дизъюнкцию).

Рассмотрим эти преобразования на примере построения минимизированной ДНФ случайной булевой функции шести переменных со степенью неопределенности 16/32, представленной картой Карно (рис. 1).



		0		1	0		1
				0		1	
1		1	1	0	0		
					0		
	1			0	0	1	
0			0				
		0		1	1		0
	1	0	1			1	

Рис. 1. Частичная булева функция шести переменных (карта Карно)

Результат работы алгоритма представлен тремя матрицами: матрицей ***U*** задана ДНФ, полученная итеративным алгоритмом, матрицей ***V*** – результат упрощения импликант путем удаления избыточных символов, матрицей ***W*** – результат упрощения ДНФ в целом путем удаления избыточных членов (отмеченных снизу звездочкой). Множества импликант, представляемых столбцами этих матриц, будем обозначать соответственно как U, V и W, а мощности этих множеств – как u, v и w:

	U	***V***	***W***
a:	01000011	----001-	---01
b:	000-110-	--0-----	-0---
c:	10001110	-000-11-	-0011
d:	1011001-	-0--001-	---01
e:	01-0-110	0--0--10	0-0-1
f:	10000--1	100-0--1	10---
		* * *	

В традиционном виде полученная ДНФ выглядит следующим образом:

$$\bar{e}\, f \vee \bar{b}\, \bar{c}\, \bar{f} \vee \bar{c}\, \bar{e} \vee \bar{a}\, c\, \bar{d} \vee a\, c\, d\, e\,.$$

В данном примере ограничимся рассмотрением довольно простой булевой функции шести переменных. В более сложной ситуации (при $n = 24$) формула полученной ДНФ обладала бы длиной порядка 100 км, а более компактное матричное представление занимало бы около 200 страниц.

Если переменных свыше 20, число v представленных в матрице ***V*** импликант может измеряться миллионами и непосредственное получение для каждой из них дизъюнкции остальных (с целью проверки возможности ее удаления) требует выполнения триллионов операций и приводит к чрезмерным затратам компьютерного времени T. В связи с этим для решения данной задачи предлагается оригинальный алгоритм, существенно более эффективный, что подтверждается экспериментом (табл. 1).

Таблица 1

Оценки времени работы программно реализованных алгоритмов

Число переменных n	12	13	14	15	16	17	…
Время работы простого алгоритма, с	0,41	5,45	39,80	294,13	2243,92	41966,07	…
Время работы нового алгоритма, с	0,21	0,44	0,96	2,57	6,94	16,64	…

8. Эффективный алгоритм удаления поглощаемых импликант

Описываемый ниже алгоритм основан на последовательном разложении упорядоченного множества импликант $V = (v_1, v_2, v_3, \ldots)$. Множество импликант V разбивается на несколько частей. Каждая из них делится на более мелкие части и т. д., пока очередные элементы разложения не будут содержать по одной импликанте. Результат этого процесса представляется деревом разложения множества V. Пример такого дерева показан на рис. 2, где число импликант равно 22, коэффициент разложения равен 3, число ярусов – 3.

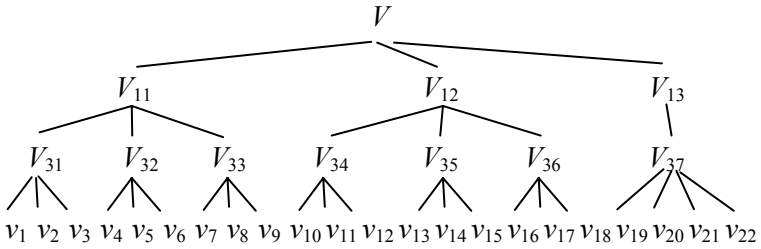


Рис. 2. Дерево разложения множества импликант

В ходе работы алгоритма последовательно рассматриваются импликанты из множества V и удаляются из этого множества, если это возможно. При анализе очередной импликанты рассматривается виртуальное дерево, образованное вершинами (они показаны темными кружками на рис. 3) на пути от рассматриваемой импликанты к корню дерева разложения. Кроме этих *основных* вершин, принадлежащих различным ярусам, в виртуальное дерево входят также *сопутствующие* им вершины – элементы разложения соседней сверху вершины.

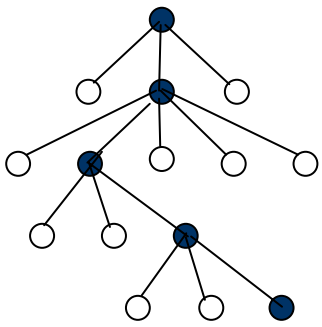


Рис. 3. Виртуальное дерево

Каждой вершине i виртуального дерева ставится в соответствие некоторая совокупность V_i импликант из множества V, а также совокупность M_i покрываемых этими импликантами элементов характеристического множества M^1. Основная вершина характеризуется также совокупностью M_i^* элементов из множества M^1, покрываемых только совокупностью V_i.

Если $M_i^* = \varnothing$, то все образующие множество V_i импликанты могут считаться избыточными и могут быть удалены из множества V. Обычно такая возможность предоставляется на нижнем ярусе и импликанты удаляются поодиночке.

Работа алгоритма начинается с верхнего яруса и заключается в обходе дерева разложения множества V вглубь, в ходе которого находятся и удаляются некоторые импликанты и корректируются содержащие их части, а также изменяется виртуальное дерево. Эффективность алгоритма

сильно зависит от выбора подходящего коэффициента разложения рассматриваемых частей и соответственно от числа уровней в дереве разложения. Аналитические и экспериментальные исследования показали, что достаточно эффективным оказался коэффициент разложения, равный 8.

9. Программная реализация и компьютерные эксперименты

Разработанный и реализованный программно (в языке C++) алгоритм ориентирован на минимизацию булевых функций с числом переменных до 24 и обладает следующими характеристиками: число ярусов $m = 6$, коэффициент разложения k на четвертом ярусе (нижнем) равен $2^2 = 4$, на третьем ярусе – $2^3 = 8$, на следующих сверху ярусах – $2^5 = 32$. Поскольку число импликант в множестве V может изменяться в широких пределах, этот коэффициент соответственно варьируется для конкретных участков.

Ниже приводятся результаты выполненных на ноутбуке HP1067 (3 ГГц) экспериментов с разработанной программой, в ходе которых рассматривались случайные булевы функции со следующими параметрами: n – число переменных, r – плотность единиц, s – степень неопределенности. Результаты проведенной минимизации представлены следующими величинами:

h – мощность характеристического множества M^1;

u – число импликант, найденных итеративным алгоритмом;

It – число итераций;

w – число импликант в полученной безызбыточной ДНФ;

S – вес этой ДНФ (сумма рангов импликант);

T – время выполнения программы в секундах.

Эффективность программы и границы ее применимости характеризуются следующими данными (табл. 2). Из табл. 2 видно, что время счета T быстро растет, увеличиваясь примерно в четыре раза с возрастанием числа аргументов на единицу. Это увеличение объясняется удвоением числа рассматриваемых импликант и удвоением длины векторов, задающих булеву функцию.

Таблица 2

Результаты минимизации для случайных булевых функций, полученными при постоянных значениях параметров $r = 16$ и $s = 16$

n	h	u	It	w	S	T
17	32 970	14 923	3	9 598	128 064	3
18	65 529	29 259	3	18 797	269 037	11
19	131 537	57 868	3	37 121	566 813	46
20	262 310	113 884	4	73 087	1 186 192	177
21	524 868	224 352	4	144 053	2 477 259	682
22	1 047 488	440 862	4	283 143	5 141 821	2 721
23	2 098 604	870 662	4	560 511	10 723 381	11 223

В табл. 3 представлено распределение импликант в полученных ДНФ по рангам, например, пара 15 – 83 означает, что в ДНФ входят 83 импликанты ранга 15.

Таблица 3

Результаты экспериментов по количеству найденных импликант

n	Распределение импликант по рангам				
17	15 – 83,	14 – 3 466,	13 – 5 706,	12 – 341	
18	16 – 106,	15 – 6 380,	14 – 11 596,	13 – 712	
19	17 – 142,	16 – 11 440,	15 – 23 811,	14 – 1 729	
20	18 – 230,	17 – 20 121,	16 – 48 954,	15 – 3 772,	14 – 5
21	19 – 315,	18 – 36 274,	17 – 98 753,	16 – 8 709,	15 – 6
22	20 – 429,	19 – 63 713,	18 – 199 679,	17 – 19 310,	16 – 8
23	21 – 558,	20 – 114 558,	19 – 403 424,	18 – 41 943,	17 – 28

Данные из табл. 4 показывают, как с ростом неопределенности функции растет эффективность операции удаления избыточных импликант из ДНФ, полученной итеративным алгоритмом: число импликант сокращается в *u/w* раз. Рассматривается пример частично определенной случайной булевой функции от 12 переменных.

Таблица 4

Зависимость эффективности операции удаления избыточных импликант от неопределенности функции

S	0	3	6	9	12	15	18	21	24	27	30
H	2049	1847	1665	1449	1290	1076	905	698	496	340	142
U	733	688	661	616	587	529	479	415	325	244	117
W	666	591	541	461	417	361	287	239	172	117	49
S	6587	5710	5125	4208	3767	3162	2409	1925	1311	810	273
u/w	1,10	1,16	1,22	1,34	1,41	1,47	1,67	1,74	1,89	2,09	2,39

Заключение

В работе предложен новый метод минимизации произвольных частичных булевых функций многих переменных. В его основе лежит оригинальная технология параллельных операций над соседними элементами в булевом пространстве многих переменных, позволяющая быстро находить в характеристическом множестве функции (где она принимает значение 1) элементы с малым числом соседей и определяемые ими импликанты минимизируемой булевой функции. Итеративное применение соответствующей процедуры к последовательно сокращаемому характеристическому множеству приводит к получению системы импликант, покрывающих функцию. Затем импликанты приводятся к простым и те из них, которые покрываются совокупностью остающихся, удаляются, в результате чего находится безызбыточная ДНФ, близкая к кратчайшей и минимальной и принимаемая за решение. Результаты компьютерных экспериментов свидетельствуют о высокой эффективности метода: при числе переменных до 20 функция минимизируется за минуту, на минимизацию функции 24 переменных затрачивается 14 ч.

Программная реализация предложенного метода включена в качестве альтернативного алгоритма минимизации булевых функций в классе ДНФ из программного комплекса для решения задач логического проектирования LOGIC-2 [10].

Список литературы

1. Quine, W.V. The problem of simplifying of truth functions / W.V. Quine // Am. Math. Monthly. – 1952. – Vol. 59, № 8. – P. 521–531.

2. McCluskey, E.J. Minimization of Boolean functions / E.J. McCluskey // Bell System Tech. J. – 1956. – Vol. 35, № 6. – P. 1417–1444.

3. Закревский, А.Д. Логический синтез каскадных схем / А.Д. Закревский. – М. : Наука, 1981. – 415 с.

4. Zakrevskij, A.D. Minimization of Boolean functions of many variables – iterative algorithm / A.D. Zakrevskij, N.R. Toropov // Internat. Workshop on Boolean Problems. – Freiberg, Germany, 2008. – P. 175–182.

5. Закревский, А.Д. Векторный метод минимизации булевых функций многих переменных / А. Д. Закревский // Доклады НАН Беларуси. – 2009. – Т. 53, № 2. – С. 45–48.

6. Закревский, А.Д. Минимизация булевых функций многих переменных в классе ДНФ – итеративный метод и программная реализация / А.Д. Закревский, Н.Р. Торопов // Прикладная дискретная математика. – 2009. – № 1 (3). – С. 5–14.

7. Закревский, А.Д. Реализация случайных событий с заданной вероятностью / А.Д. Закревский // Труды СФТИ. – 1965. – Вып. 47. – С. 56–59.

8. Zakrevskij, A.D. Parallel operations over neighbors in Boolean space / A.D. Zakrevskij // Proceedings of the Sixth International Conference CAD DD-07. – Minsk, 2007. – Vol. 2. – P. 6–13.

9. Закревский, А.Д. Программирование вычислений в многомерном булевом пространстве / А.Д. Закревский // Управление, вычислительная техника и информатика. Вестник Томского государственного университета. – 2008. – № 2 (3). – С. 94–105.

10. Романов, В.И. Программный комплекс для автоматизации исследований комбинаторных алгоритмов логического проектирования / В.И. Романов // Информационные системы и технологии» (IST'2009) : материалы V Междунар. конф.-форума, Минск, 16–17 ноября 2009 г. В 2 ч. Ч. 1 ; редкол. : Н.И. Листопадов [и др.]. – Минск : А.Н. Вараксин, 2009. – С. 95–98.

Поступила 30.09.09

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail: Zakrevskij@tut.by

A.D. Zakrevskij, N.R. Toropov, V.I. Romanov

DNF-IMPLEMENTATION OF PARTIAL BOOLEAN FUNCTIONS OF MANY VARIABLES

A heuristic method for minimization of partial Boolean functions of many variables is suggested, oriented towards computer implementation. It is based on the original technology of parallel operations over neighbor elements in Boolean space of many variables. The method was implemented in software, and computer experiments confirmed its efficiency.